Monet CycleGAN

This project explores generating "Monet Style" images from photos implementing Cycle-Consistent Adversarial Network (CycleGAN). At the time of writing, this project is an assignment for "DTSA 5511: Introduction to Deep Learning". This is my first time implementing CycleGAN or working with TensorFlow Record (tfrec) files so I relied upon many sources and tutorials to complete this project. The main difference between this model from source documentation and tutorials is additional jittering steps and the implementation of an efficient neural network (ENet) as opposed to UNet or Resnet to construct the generator. I have included a comprehensive list of sources at the end of the notebook that I found helpful throughout the project.

This notebook can be accessed from my github repository

You can read more about the kaggle competition and access the original data files

In [12]:

```
#Import neccesary libraries and packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential
from keras.layers import Input, Add, Rescaling, Concatenate, ZeroPadding2D, Conv2D, MaxPool2D, Conv2DTranspose,
UpSampling2D, Dropout, Flatten, GroupNormalization, BatchNormalization, Activation, LeakyReLU
import tensorflow as tf
import os
import time
import PIL
import shutil
#for dirname, _, filenames in os.walk('/kaggle/input'):
    #for filename in filenames:
        #print(os.path.join(dirname, filename))
```

EDA and Preprocessing

We first need to access the images from the directory. In total, there are 300 monet images and 7028 photo images.

In [13]:
```
#Get all the file names
monet_files = tf.io.gfile.glob(str('/kaggle/input/gan-getting-started/monet_tfrec/*.tfrec'))
photo_files = tf.io.gfile.glob(str('/kaggle/input/gan-getting-started/photo_tfrec/*.tfrec'))
```

Now we need to decode and read in the images. We will also normalize the images so that colors on the higher end of the RGB spectrum do not have a heavier impact on weights than colors on the lower end of the RGB spectrum. Lastly, we will apply some random jittering to introduce some variation in the dataset and prevent overfitting.There are mutliple ways to do this,but we will stick to taking a random crop of the original image and radomly flipping the image vertically or horizontally. In some scenarios it might be useful to also randomly adjust the hue, saturation, or contrast of the image, but that would not be appropriate here since these are features that often help distinguish an artists paintings.

One nice thing about TensorFlow is all of these functions can be wrapped into the data pipeline and applied to each batch with only a few lines of code.

In [14]:

```
#Implement helper functions to decode and preprocess the images
def preprocess_tfrecord(example,jitter=True):
    tfrecord_format = {
        "image_name": tf.io.FixedLenFeature([], tf.string),
        "image": tf.io.FixedLenFeature([], tf.string),
        "target": tf.io.FixedLenFeature([], tf.string)
    }
    example = tf.io.parse_single_example(example, tfrecord_format)
    #decode
    image = tf.image.decode_jpeg(example['image'], channels=3)
    #normalize so colors on the upper end of the RGB spectrum are not washed out
    image = (tf.cast(image, tf.float32)/ 127.5) - 1
    #jitter
    if jitter:
        image=tf.image.resize(image, size=[286,286])
        image=tf.image.random_crop(image, size=[256,256,3])
        image=tf.image.random_flip_left_right(image)
        image=tf.image.random_flip_up_down(image)
        return image

#implement helper function to load the datasets
def load_dataset(filenames, labeled=True, ordered=False):
    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.map(preprocess_tfrecord, num_parallel_calls=tf.data.AUTOTUNE)
    return dataset

#Create the data pipeline
monets = load_dataset(monet_files, labeled=True).shuffle(300).batch(1)
photos = load_dataset(photo_files, labeled=True).shuffle(300).batch(1)
```

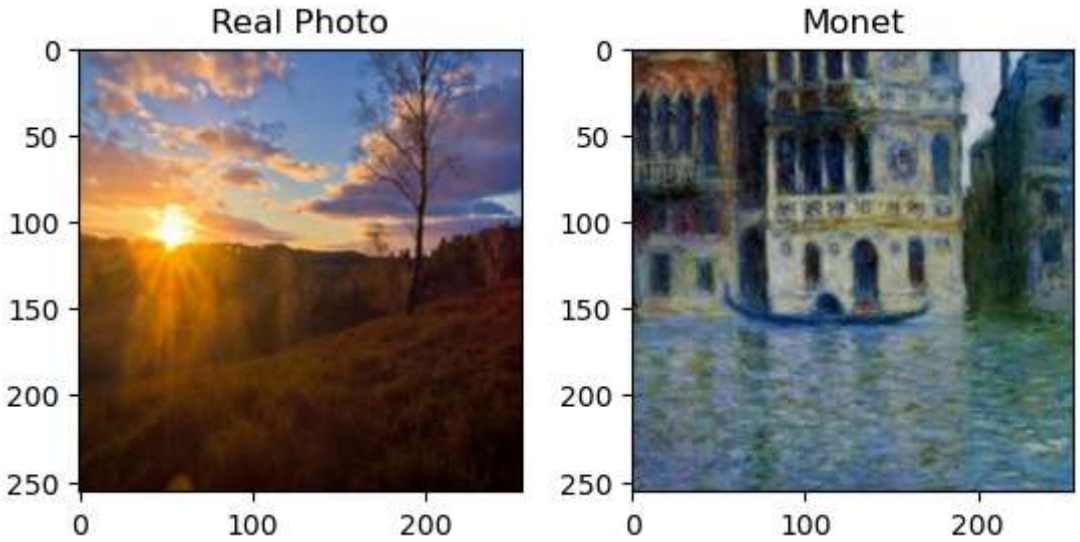
Now that we have imported the data lets take a look at what a few images from each dataset look like.

In [15]:

```
#using next(iter()) is a convenient way to iterate through a dataset of TensorFlow object
ex_monet=next(iter(monets))
ex_photo=next(iter(photos))

#Plot some examples
#Create a subplot with 2 examples
plt.subplot(121)
plt.subplots_adjust(wspace=0.25)
plt.title('Real Photo')
#(note: * 0.5 + 0.5 operation is required to normalize the image so imshow can read it properly)
plt.imshow(ex_photo[0] * 0.5 + 0.5)
plt.subplot(122)
plt.title('Monet')
plt.imshow(ex_monet[0] * 0.5 + 0.5)
```

Out[15]:

```
<matplotlib.image.AxesImage at 0x78f1c014c7c0>
```



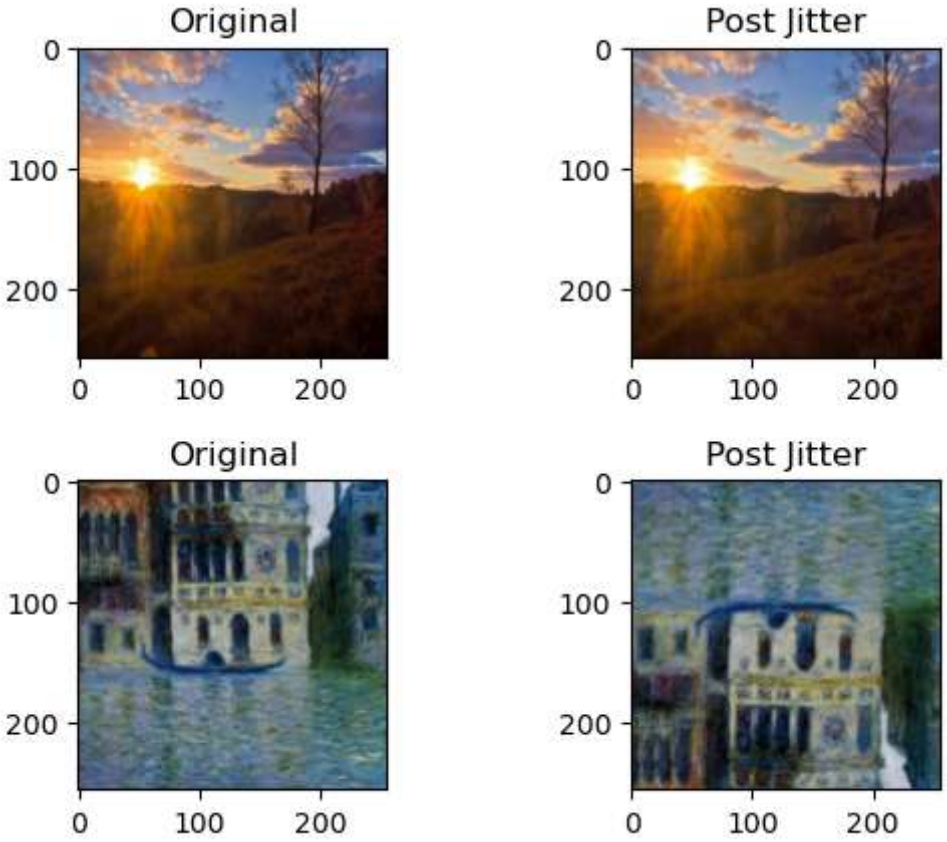
Random jittering is wrapped into the model, but lets also use our examples to visualize what is happening.

In [16]:

```
#Example of jitter preprocessing
def jitter(image):
    image=tf.image.resize(image, size=[286,286])
    image=tf.image.random_crop(image, size=[256,256,3])
    image=tf.image.random_flip_left_right(image)
    image=tf.image.random_flip_up_down(image)
    return image

ex_monet_new=jitter(ex_monet[0])
ex_photo_new=jitter(ex_photo[0])

#Create a before and after subplot
plt.subplot(221)
plt.subplots_adjust(wspace=0.25, hspace=0.4)
plt.title('Original')
#(note: * 0.5 + 0.5 operation is required to normalize the image so imshow can read it properly)
plt.imshow(ex_photo[0] * 0.5 + 0.5)
plt.subplot(222)
plt.title('Post Jitter')
#(note: * 0.5 + 0.5 operation is required to normalize the image so imshow can read it properly)
plt.imshow(ex_photo_new * 0.5 + 0.5)
plt.subplot(223)
plt.title('Original')
plt.imshow(ex_monet[0] * 0.5 + 0.5)
plt.subplot(224)
plt.title('Post Jitter')
plt.imshow(ex_monet_new * 0.5 + 0.5)
```

Out[16]:

```
<matplotlib.image.AxesImage at 0x78f1c008cfa0>
```



Model Building

Now we need to construct the actual model. One interesting feature of CycleGAN is that images from each class are unpaired, meaning we are picking up on general trends/styles and not linking two specific images together. This is not only more flexible than supervised approaches but also less costly as the need to identify and assign image pairs is unnecessary.

They also differ from vanilla GAN and DCGAN in a few ways, but the primary difference is the cyclical approach to evaluating the model. More specifically, the model does not simply measure the adversarial loss of the generated image, but the loss of the generated image translated back to its original class. This concept has been referred to as "Cycle Consistency Loss". To accomplish this the CycleGAN architecture has 2 discriminators and 2 generators that opperate in a cyclical manner, hence the name.The general cycle goes like this:

1. an image from class a is passed through a generator to be transformed to an image of class b
2. the class b discriminator makes a prediction if the image is of class b
3. the transformed image is passed into another generator and transformed back to class a
4. the difference between the original image and the regenerated image is measured and weights are updated

Note: This is a gross oversimplification of the process, but should give you enough of an overview to interperet what is going on in the code below. If you need more details the full research paper that is linked in the sources section.

Construct The Generators

There are a couple popular generator architechtures used for CycleGAN, but most papers/tutorial use some form of unet or resnet. Both of these methods are computationally expensive. For this project we will attempt to implement Enet which has been show to produce similar results in image segmentation tasks with smaller computational and memory requirements.

In [17]:

```
#Define Enccoder
def bn_encode(filters,
              ds=False,#case for downsampling
              asym=False,#case for asymmetric convolution
              dial=(1,1),#case for dialated convolution
              drate=0.1,
              dial_rate=1,
              apply_norm=True):

    filters=int(filters)
    initializer = tf.random_normal_initializer(0., 0.02)
    gamma_init = keras.initializers.RandomNormal(mean=0.0, stddev=0.02)
    result = Sequential()

    #Initial Projection
    if ds:
        result.add(Conv2D(filters, kernel_size=2, strides=2, padding='same',
                          kernel_initializer=initializer, use_bias=False))
    else:
        result.add(Conv2D(filters/2, kernel_size=1, strides=1, padding='same',
                          kernel_initializer=initializer, use_bias=False))
    if apply_norm:
        result.add(BatchNormalization(gamma_initializer=gamma_init))
        #result.add(GroupNormalization(groups=filters/2,gamma_initializer=gamma_init))

    result.add(LeakyReLU())
    #Main Convolution
    if asym:
        result.add(Conv2D(filters/2, kernel_size=(5,1), strides=1, padding='same',
                          kernel_initializer=initializer, use_bias=False))
        result.add(Conv2D(filters/2, kernel_size=(1,5), strides=1, padding='same',
                          kernel_initializer=initializer, use_bias=False))
    else:
```

```
        result.add(Conv2D(filters/2, kernel_size=3, strides=1, padding='same',
                          kernel_initializer=initializer, use_bias=False))
    if apply_norm:
        result.add(BatchNormalization(gamma_initializer=gamma_init))
        #result.add(GroupNormalization(groups=filters/2,gamma_initializer=gamma_init))

    result.add(LeakyReLU())

    #Final Expansion
    result.add(Conv2D(filters, kernel_size=1, strides=1, padding='same',
                      kernel_initializer=initializer, use_bias=False))
    if apply_norm:
        result.add(BatchNormalization(gamma_initializer=gamma_init))
        #result.add(GroupNormalization(groups=filters, gamma_initializer=gamma_init))
    result.add(LeakyReLU())

    #Regulizer
    result.add(Dropout(rate=drate))
    return result
```

In [18]:

```
#Define Bottleneck Decoder
def bn_decode(filters,
              us=False,#case for upsampling
              drate=0.1,
              apply_norm=True):

    filters=int(filters)
    initializer = tf.random_normal_initializer(0., 0.02)
    gamma_init = keras.initializers.RandomNormal(mean=0.0, stddev=0.02)
    result = Sequential()

    #Initial Projection
    if us:
        result.add(Conv2DTranspose(filters, kernel_size=2, strides=2, padding='same',
                           kernel_initializer=initializer, use_bias=False))
    else:
        result.add(Conv2D(filters/2, kernel_size=1, strides=1, padding='same',
                           kernel_initializer=initializer, use_bias=False))
    if apply_norm:
        result.add(BatchNormalization(gamma_initializer=gamma_init))
        #result.add(GroupNormalization(groups=filters/2,gamma_initializer=gamma_init))

    result.add(LeakyReLU())
    #Main Convolution
    result.add(Conv2D(filters/2, kernel_size=3, strides=1,padding='same',
                      kernel_initializer=initializer, use_bias=False))
    if apply_norm:
        result.add(BatchNormalization(gamma_initializer=gamma_init))
        #result.add(GroupNormalization(groups=filters/2,gamma_initializer=gamma_init))

    result.add(LeakyReLU())

    #Final Expansion
```

```
    result.add(Conv2D(filters, kernel_size=1, strides=1, padding='same',
                      kernel_initializer=initializer, use_bias=False))
    if apply_norm:
        result.add(BatchNormalization(gamma_initializer=gamma_init))
        #result.add(GroupNormalization(groups=filters, gamma_initializer=gamma_init))
    result.add(LeakyReLU())

    #Regulizer
    result.add(Dropout(rate=drate))
    return result
```

Now it is time to create the generator following the architechture described in the Enet for image segmentation paper

In [19]:

```
#Define Generator
def Generator():
    initializer = tf.random_normal_initializer(0., 0.02)
    gamma_init = keras.initializers.RandomNormal(mean=0.0, stddev=0.02)
    inputs = Input(shape=[256,256,3], batch_size=1)

    #initial Enet block
    #output 16x128x128
    I1=Conv2D(13,kernel_size=3,strides=2, padding='same',
              kernel_initializer=initializer, use_bias=False)(inputs)
    I2=MaxPool2D(pool_size=(2, 2))(inputs)
    I=Concatenate()([I1, I2])
    #stage 1 (outputs 64x64x64)
    #b1
    b1=bn_encode(filters=64,ds=True,drate=0.01)(I)
    paddings = [(0, 0), (0,0), (0, 0), (48, 0)]
    I_padded = tf.pad(I, paddings, mode="constant")
    b1f=Add()([MaxPool2D(pool_size=(2, 2))(I_padded),b1])
    #b11
    b11=bn_encode(filters=64,drate=0.01)(b1f)
    b11f=Add()([b1f,b11])
    #b12
    b12=bn_encode(filters=64,drate=0.01)(b11f)
    b12f=Add()([b11f,b12])
    #b13
    b13=bn_encode(filters=64,drate=0.01)(b12f)
    b13f=Add()([b12f,b13])
    #b14
    b14=bn_encode(filters=64,drate=0.01)(b13f)
    b14f=Add()([b13f,b14])

    #stage 2 (ouputs 128x32x32)
    #b2
```

```
b2=bn_encode(filters=128,ds=True)(b14f)
paddings = [(0, 0), (0,0), (0, 0), (64, 0)]
b14f_padded = tf.pad(b14f, paddings, mode="constant")
b2f=Add()([MaxPool2D(pool_size=(2, 2))(b14f_padded),b2])
#b21
b21=bn_encode(filters=128)(b2f)
b21f=Add()([b2f,b21])
#b22
b22=bn_encode(filters=128, dial_rate=(2,2))(b21f)
b22f=Add()([b21f,b22])
#b23
b23=bn_encode(filters=128, asym=True)(b22f)
b23f=Add()([b22f,b23])
#b24
b24=bn_encode(filters=128, dial_rate=(4,4))(b23f)
b24f=Add()([b23f,b24])
#b25
b25=bn_encode(filters=128)(b24f)
b25f=Add()([b24f,b25])
#b26
b26=bn_encode(filters=128, dial_rate=(8,8))(b25f)
b26f=Add()([b25f,b26])
#b27
b27=bn_encode(filters=128, asym=True)(b26f)
b27f=Add()([b26f,b27])
#b28
b28=bn_encode(filters=128, dial_rate=(16,16))(b27f)
b28f=Add()([b27f,b28])

#stage 3 (ouputs 128x32x32)
#b31
b31=bn_encode(filters=128)(b28f)
b31f=Add()([b28f,b31])
#b32
```

```
b32=bn_encode(filters=128, dial_rate=(2,2))(b31f)
b32f=Add()([b31f,b32])
#b33
b33=bn_encode(filters=128, asym=True)(b32f)
b33f=Add()([b32f,b33])
#b34
b34=bn_encode(filters=128, dial_rate=(4,4))(b33f)
b34f=Add()([b33f,b34])
#b35
b35=bn_encode(filters=128)(b34f)
b35f=Add()([b34f,b35])
#b36
b36=bn_encode(filters=128, dial_rate=(8,8))(b35f)
b36f=Add()([b35f,b36])
#b37
b37=bn_encode(filters=128, asym=True)(b36f)
b37f=Add()([b36f,b37])
#b38
b38=bn_encode(filters=128, dial_rate=(16,16))(b37f)
b38f=Add()([b37f,b38])

#stage 4 (outpust 64x64x64)
#b4
b4=bn_decode(filters=64,us=True)(b38f)
#b41
b41=bn_decode(filters=64)(b4)
b41f=Add()([b4,b41])
#b42
b42=bn_decode(filters=64)(b41f)
b42f=Add()([b41f,b42])
#stage 5 (outputs16x128x128)
#b5
b5=bn_decode(filters=16,us=True)(b42f)
#b51
```

```
    b51=bn_decode(filters=16)(b5)
    b51f=Add()([b5,b51])
    #Upscale to original image size
    last = Conv2DTranspose(3, 4,strides=2, padding='same',
                                kernel_initializer=initializer,
                                activation='tanh') # (bs, 256, 256, 3)
    x = last(b51f)

    return keras.Model(inputs=inputs, outputs=x)

sample_generator=Generator()
sample_generator.summary()
```

```
/opt/conda/lib/python3.10/site-packages/keras/initializers/initializers.py:120: UserWarning: The initializer RandomNormal is unseeded and being called multiple times, which will return identical values each time (even if the initializer is unseeded). Please update your code to provide a seed to the initializer, or avoid using the same initalizer instance more than once.
  warnings.warn(
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(1, 256, 256, 3)]   0           []

 conv2d (Conv2D)                (1, 128, 128, 13)    351         ['input_1[0][0]']

 max_pooling2d (MaxPooling2D)   (1, 128, 128, 3)     0           ['input_1[0][0]']

 concatenate (Concatenate)      (1, 128, 128, 16)    0           ['conv2d[0][0]',
                                                                  'max_pooling2d[0][0]']

 tf.compat.v1.pad (TFOpLambda)  (1, 128, 128, 64)    0           ['concatenate[0][0]']

 max_pooling2d_1 (MaxPooling2D)  (1, 64, 64, 64)     0           ['tf.compat.v1.pad[0][0]']

 sequential (Sequential)        (1, 64, 64, 64)      25216       ['concatenate[0][0]']

 add (Add)                      (1, 64, 64, 64)      0           ['max_pooling2d_1[0][0]',
                                                                  'sequential[0][0]']

 sequential_1 (Sequential)      (1, 64, 64, 64)      13824       ['add[0][0]']

 add_1 (Add)                    (1, 64, 64, 64)      0           ['add[0][0]',
                                                                  'sequential_1[0][0]']

 sequential_2 (Sequential)      (1, 64, 64, 64)      13824       ['add_1[0][0]']

 add_2 (Add)                    (1, 64, 64, 64)      0           ['add_1[0][0]',
                                                                  'sequential_2[0][0]']

 sequential_3 (Sequential)      (1, 64, 64, 64)      13824       ['add_2[0][0]']
```

```
add_3 (Add)                    (1, 64, 64, 64)      0           ['add_2[0][0]',
                                                                 'sequential_3[0][0]']

sequential_4 (Sequential)      (1, 64, 64, 64)      13824       ['add_3[0][0]']

add_4 (Add)                    (1, 64, 64, 64)      0           ['add_3[0][0]',
                                                                 'sequential_4[0][0]']

tf.compat.v1.pad_1 (TFOpLambda  (1, 64, 64, 128)    0           ['add_4[0][0]']
)

max_pooling2d_2 (MaxPooling2D)  (1, 32, 32, 128)    0           ['tf.compat.v1.pad_1[0][0]']

sequential_5 (Sequential)      (1, 32, 32, 128)     115968      ['add_4[0][0]']

add_5 (Add)                    (1, 32, 32, 128)     0           ['max_pooling2d_2[0][0]',
                                                                 'sequential_5[0][0]']

sequential_6 (Sequential)      (1, 32, 32, 128)     54272       ['add_5[0][0]']

add_6 (Add)                    (1, 32, 32, 128)     0           ['add_5[0][0]',
                                                                 'sequential_6[0][0]']

sequential_7 (Sequential)      (1, 32, 32, 128)     54272       ['add_6[0][0]']

add_7 (Add)                    (1, 32, 32, 128)     0           ['add_6[0][0]',
                                                                 'sequential_7[0][0]']

sequential_8 (Sequential)      (1, 32, 32, 128)     58368       ['add_7[0][0]']

add_8 (Add)                    (1, 32, 32, 128)     0           ['add_7[0][0]',
                                                                 'sequential_8[0][0]']
```

```
sequential_9 (Sequential)      (1, 32, 32, 128)     54272       ['add_8[0][0]']

add_9 (Add)                    (1, 32, 32, 128)     0           ['add_8[0][0]',
                                                                 'sequential_9[0][0]']

sequential_10 (Sequential)     (1, 32, 32, 128)     54272       ['add_9[0][0]']

add_10 (Add)                   (1, 32, 32, 128)     0           ['add_9[0][0]',
                                                                 'sequential_10[0][0]']

sequential_11 (Sequential)     (1, 32, 32, 128)     54272       ['add_10[0][0]']

add_11 (Add)                   (1, 32, 32, 128)     0           ['add_10[0][0]',
                                                                 'sequential_11[0][0]']

sequential_12 (Sequential)     (1, 32, 32, 128)     58368       ['add_11[0][0]']

add_12 (Add)                   (1, 32, 32, 128)     0           ['add_11[0][0]',
                                                                 'sequential_12[0][0]']

sequential_13 (Sequential)     (1, 32, 32, 128)     54272       ['add_12[0][0]']

add_13 (Add)                   (1, 32, 32, 128)     0           ['add_12[0][0]',
                                                                 'sequential_13[0][0]']

sequential_14 (Sequential)     (1, 32, 32, 128)     54272       ['add_13[0][0]']

add_14 (Add)                   (1, 32, 32, 128)     0           ['add_13[0][0]',
                                                                 'sequential_14[0][0]']

sequential_15 (Sequential)     (1, 32, 32, 128)     54272       ['add_14[0][0]']

add_15 (Add)                   (1, 32, 32, 128)     0           ['add_14[0][0]',
                                                                 'sequential_15[0][0]']
```

```
sequential_16 (Sequential)     (1, 32, 32, 128)     58368       ['add_15[0][0]']

add_16 (Add)                   (1, 32, 32, 128)     0           ['add_15[0][0]',
                                                                 'sequential_16[0][0]']

sequential_17 (Sequential)     (1, 32, 32, 128)     54272       ['add_16[0][0]']

add_17 (Add)                   (1, 32, 32, 128)     0           ['add_16[0][0]',
                                                                 'sequential_17[0][0]']

sequential_18 (Sequential)     (1, 32, 32, 128)     54272       ['add_17[0][0]']

add_18 (Add)                   (1, 32, 32, 128)     0           ['add_17[0][0]',
                                                                 'sequential_18[0][0]']

sequential_19 (Sequential)     (1, 32, 32, 128)     54272       ['add_18[0][0]']

add_19 (Add)                   (1, 32, 32, 128)     0           ['add_18[0][0]',
                                                                 'sequential_19[0][0]']

sequential_20 (Sequential)     (1, 32, 32, 128)     58368       ['add_19[0][0]']

add_20 (Add)                   (1, 32, 32, 128)     0           ['add_19[0][0]',
                                                                 'sequential_20[0][0]']

sequential_21 (Sequential)     (1, 32, 32, 128)     54272       ['add_20[0][0]']

add_21 (Add)                   (1, 32, 32, 128)     0           ['add_20[0][0]',
                                                                 'sequential_21[0][0]']

sequential_22 (Sequential)     (1, 64, 64, 64)      53888       ['add_21[0][0]']

sequential_23 (Sequential)     (1, 64, 64, 64)      13824       ['sequential_22[0][0]']
```

```
 add_22 (Add)                   (1, 64, 64, 64)      0           ['sequential_22[0][0]',
                                                                  'sequential_23[0][0]']

 sequential_24 (Sequential)     (1, 64, 64, 64)      13824       ['add_22[0][0]']

 add_23 (Add)                   (1, 64, 64, 64)      0           ['add_22[0][0]',
                                                                  'sequential_24[0][0]']

 sequential_25 (Sequential)     (1, 128, 128, 16)    5536        ['add_23[0][0]']

 sequential_26 (Sequential)     (1, 128, 128, 16)    960         ['sequential_25[0][0]']

 add_24 (Add)                   (1, 128, 128, 16)    0           ['sequential_25[0][0]',
                                                                  'sequential_26[0][0]']

 conv2d_transpose_2 (Conv2DTran  (1, 256, 256, 3)    771         ['add_24[0][0]']
 spose)

==================================================================================================
Total params: 1,170,370
Trainable params: 1,159,218
Non-trainable params: 11,152
__________________________________________________________________________________________________
```

In [20]:

```
#generator to transform images to photo style
generator_p=Generator()
#generator to transfrom images to monet style
generator_m=Generator()
```

Construct the Discriminators

The discriminators will have a PatchGAN architechture. Since considerable time was used implementing ENet when building the generator, we will not reinvent the wheel with the discriminator. We will implement the discriminator avaliable in the tensorflow_examples package. Since the package is not downloadable from the kaggle notebook, we will pull the helper functions from the pix2pix tutorial and make minimal adjustments to work with the packages that have already been imported. Documentation for the package can be found in the sources section, but here is a quote from the documentaion on how it works.

- Each block in the discriminator is: Convolution → Batch normalization → Leaky ReLU.
- The shape of the output after the last layer is (batch_size, 30, 30, 1).
- Each 30 × 30 image patch of the output classifies a 70 × 70 portion of the input image.
- The discriminator receives 2 inputs:
  - The input image and the target image, which it should classify as real.
  - The input image and the generated image (the output of the generator), which it should classify as fake.
    - The 2 imputs are concatenated together an evaluated

In [21]:

```
#Implement downsample and discriminator helper functions from
#https://www.tensorflow.org/tutorials/generative/pix2pix

def downsample(filters, size, apply_batchnorm=True):
    initializer = tf.random_normal_initializer(0., 0.02)
    result = Sequential()
    result.add(Conv2D(filters, size, strides=2, padding='same',
                      kernel_initializer=initializer, use_bias=False))
    if apply_batchnorm:
        result.add(BatchNormalization())
    result.add(LeakyReLU())
    return result

def Discriminator():
    initializer = tf.random_normal_initializer(0., 0.02)
    inp = Input(shape=[256, 256, 3], name='input_image',batch_size=1)
    down1 = downsample(64, 4, False)(inp)  # (batch_size, 128, 128, 64)
    down2 = downsample(128, 4)(down1)  # (batch_size, 64, 64, 128)
    down3 = downsample(256, 4)(down2)  # (batch_size, 32, 32, 256)
    zero_pad1 = ZeroPadding2D()(down3)  # (batch_size, 34, 34, 256)
    conv = Conv2D(512, 4, strides=1,
                  kernel_initializer=initializer,
                  use_bias=False)(zero_pad1)  # (batch_size, 31, 31, 512)
    batchnorm1 = BatchNormalization()(conv)
    leaky_relu = LeakyReLU()(batchnorm1)
    zero_pad2 = ZeroPadding2D()(leaky_relu)  # (batch_size, 33, 33, 512)
    last = Conv2D(1, 4, strides=1,
                  kernel_initializer=initializer)(zero_pad2)  # (batch_size, 30, 30, 1)
    return keras.Model(inputs=inp, outputs=last)

sample_discriminator=Discriminator()
sample_discriminator.summary()
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_image (InputLayer)    [(1, 256, 256, 3)]        0

 sequential_81 (Sequential)  (1, 128, 128, 64)         3072

 sequential_82 (Sequential)  (1, 64, 64, 128)          131584

 sequential_83 (Sequential)  (1, 32, 32, 256)          525312

 zero_padding2d (ZeroPadding  (1, 34, 34, 256)         0
 2D)

 conv2d_255 (Conv2D)         (1, 31, 31, 512)          2097152

 batch_normalization_245 (Ba  (1, 31, 31, 512)         2048
 tchNormalization)

 leaky_re_lu_246 (LeakyReLU)  (1, 31, 31, 512)         0

 zero_padding2d_1 (ZeroPaddi  (1, 33, 33, 512)         0
 ng2D)

 conv2d_256 (Conv2D)         (1, 30, 30, 1)            8193

=================================================================
Total params: 2,767,361
Trainable params: 2,765,569
Non-trainable params: 1,792
_________________________________________________________________
```

In [22]:

```
#discriminator for photo style images
discriminator_p=Discriminator()
#discriminator for monet style images
discriminator_m=Discriminator()
```

Define the Loss Function

Generator Loss- How good is the generator at generating images that look real to the discriminator?

Discriminator Loss- How good is the discriminator at determining if an image is generated?

Cycle Consistency Loss-How similar is an image that is tranformed to the different style and then transformed back to the original image

Identity Loss-How similar is a generated image of the same style (ex. similarity of a monet photo passed into the monet generator)

In [23]:

```
LAMBDA = 10
loss_obj = tf.keras.losses.BinaryCrossentropy(from_logits=True, reduction=tf.keras.losses.Reduction.NONE)
#Discriminator Loss
def discriminator_loss(real, generated):
    real_loss = loss_obj(tf.ones_like(real), real)
    generated_loss = loss_obj(tf.zeros_like(generated), generated)
    total_disc_loss = real_loss + generated_loss
    return total_disc_loss * 0.5
#Generator Loss
def generator_loss(generated):
    return loss_obj(tf.ones_like(generated), generated)
#Cycle Consistency Loss
def calc_cycle_loss(real_image, cycled_image):
    loss1 = tf.reduce_mean(tf.abs(real_image - cycled_image))
    return LAMBDA * loss1
#Identity Loss
def identity_loss(real_image, same_image):
    loss = tf.reduce_mean(tf.abs(real_image - same_image))
    return LAMBDA * 0.5 * loss
```

In [24]:

```
#define optimizers
generator_m_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
generator_p_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)

discriminator_m_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
discriminator_p_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
```

Link everything together into the CycleGAN model

In [25]:

```
@tf.function
def train_step(real_photo, real_monet):
  # persistent is set to True because the tape is used more than
  # once to calculate the gradients.
    with tf.GradientTape(persistent=True) as tape:
        # Generator m translates to monet style
        # Generator p translates to photo style
        fake_monet = generator_m(real_photo, training=True)
        cycled_photo = generator_p(fake_monet, training=True)

        fake_photo = generator_p(real_monet, training=True)
        cycled_monet = generator_m(fake_photo, training=True)

        # same_photo and same_monet are used for identity loss.
        same_photo = generator_p(real_photo, training=True)
        same_monet = generator_m(real_monet, training=True)

        disc_real_monet = discriminator_m(real_monet, training=True)
        disc_real_photo = discriminator_p(real_photo, training=True)

        disc_fake_monet = discriminator_m(fake_monet, training=True)
        disc_fake_photo = discriminator_p(fake_photo, training=True)

        # calculate the loss
        gen_monet_loss = generator_loss(disc_fake_monet)
        gen_photo_loss = generator_loss(disc_fake_photo)

        total_cycle_loss = calc_cycle_loss(real_monet, cycled_monet) + calc_cycle_loss(real_photo, cycled_phot
o)
```

```
    # Total generator loss = adversarial loss + cycle loss
    total_gen_monet_loss = gen_monet_loss + total_cycle_loss + identity_loss(real_monet, same_monet)
    total_gen_photo_loss = gen_photo_loss + total_cycle_loss + identity_loss(real_photo, same_photo)

    disc_monet_loss = discriminator_loss(disc_real_monet, disc_fake_monet)
    disc_photo_loss = discriminator_loss(disc_real_photo, disc_fake_photo)

    # Calculate the gradients for generator and discriminator
    generator_m_gradients = tape.gradient(total_gen_monet_loss,
                                          generator_m.trainable_variables)
    generator_p_gradients = tape.gradient(total_gen_photo_loss,
                                          generator_p.trainable_variables)

    discriminator_m_gradients = tape.gradient(disc_monet_loss,
                                              discriminator_m.trainable_variables)
    discriminator_p_gradients = tape.gradient(disc_photo_loss,
                                              discriminator_p.trainable_variables)

    # Apply the gradients to the optimizer
generator_m_optimizer.apply_gradients(zip(generator_m_gradients,
                                          generator_m.trainable_variables))

generator_p_optimizer.apply_gradients(zip(generator_p_gradients,
                                          generator_p.trainable_variables))

discriminator_m_optimizer.apply_gradients(zip(discriminator_m_gradients,
                                              discriminator_m.trainable_variables))

discriminator_p_optimizer.apply_gradients(zip(discriminator_p_gradients,
                                              discriminator_p.trainable_variables))
```

Results and Analysis

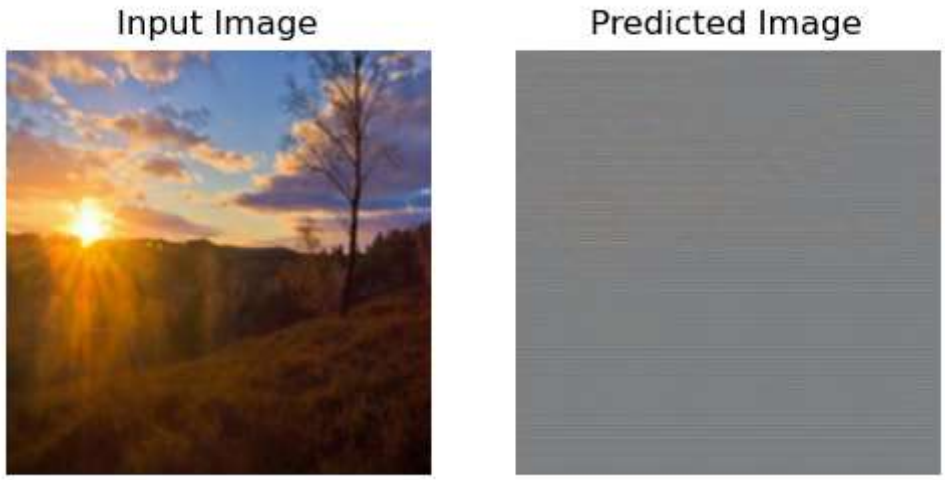
Now that we have built the model and defined the loss functions, we need to train the model. I am only using free computational resources to train the model, so the number of epochs trained will be somewhat limited.

Train the Model

In [26]:

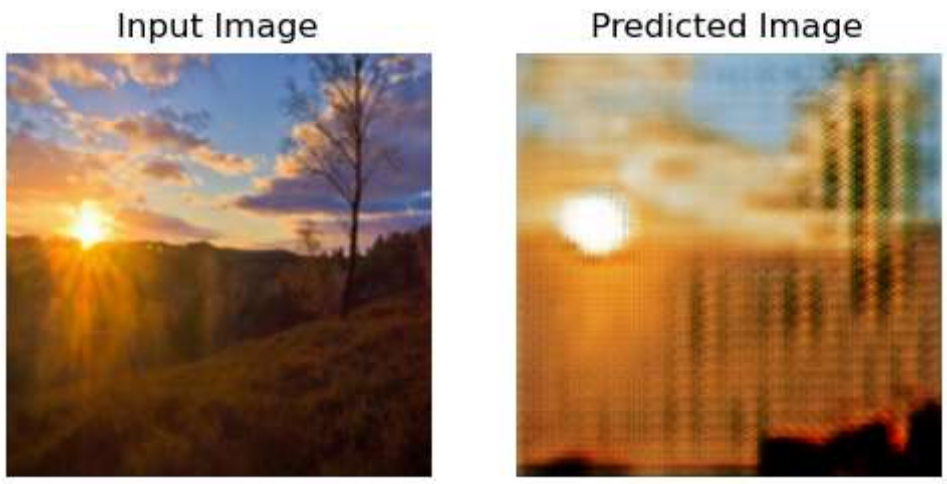
```
for epoch in range(250):
    start = time.time()
    n = 0
    for real_photo, real_monet in tf.data.Dataset.zip((photos, monets)):
        train_step(real_photo, real_monet)
        if n % 30 == 0:
            print ('.', end='')
        n += 1
    print('Time taken for epoch {} is {} sec\n'.format(epoch + 1, time.time()-start))
    if epoch % 10==0:
        prediction = generator_m(ex_photo_new[None, ...])
        plt.figure(figsize=(6,6))
        display_list = [ex_photo_new, prediction[0]]
        title = ['Input Image', 'Predicted Image']

        for i in range(2):
            plt.subplot(1, 2, i+1)
            plt.title(title[i])
            # getting the pixel values between [0, 1] to plot it.
            plt.imshow(display_list[i] * 0.5 + 0.5)
            plt.axis('off')
        plt.show()
```

```
2023-08-13 15:34:16.530351: E tensorflow/core/grappler/optimizers/meta_optimizer.cc:954] layout failed: INVALID_ARGUMENT: Size of values 0 does not match size of permutation 4 @ fanin shape inmodel_2/sequential_54/dropout_54/dropout/SelectV2-2-TransposeNHWCToNCHW-LayoutOptimizer

.........Time taken for epoch 1 is 184.94030618667603 sec
```

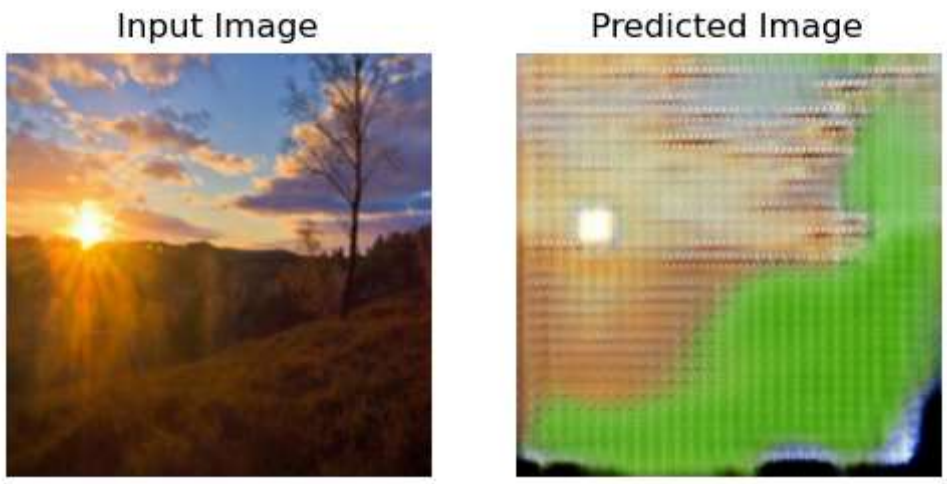


```
.....Time taken for epoch 2 is 49.3060622215271 sec

.....Time taken for epoch 3 is 49.09601044654846 sec

.....Time taken for epoch 4 is 49.21508598327637 sec

.....Time taken for epoch 5 is 49.07926297187805 sec

.....Time taken for epoch 6 is 49.90027451515198 sec

.....Time taken for epoch 7 is 50.05703282356262 sec

.....Time taken for epoch 8 is 49.83216309547424 sec

.....Time taken for epoch 9 is 49.92344570159912 sec

.....Time taken for epoch 10 is 49.57429838180542 sec

.....Time taken for epoch 11 is 49.95459461212158 sec
```

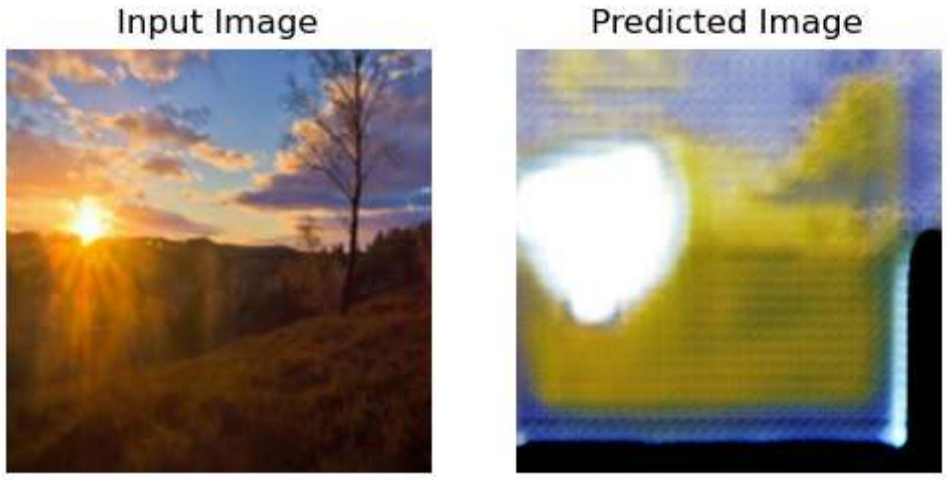
```
.....Time taken for epoch 12 is 81.93668389320374 sec

.....Time taken for epoch 13 is 49.5866494178772 sec

.....Time taken for epoch 14 is 49.542261362075806 sec

.....Time taken for epoch 15 is 49.185978412628174 sec

.....Time taken for epoch 16 is 49.47562384605408 sec

.....Time taken for epoch 17 is 49.130534172058105 sec

.....Time taken for epoch 18 is 49.468852281570435 sec

.....Time taken for epoch 19 is 49.035329818725586 sec

.....Time taken for epoch 20 is 49.521634101867676 sec

.....Time taken for epoch 21 is 49.91589117050171 sec
```



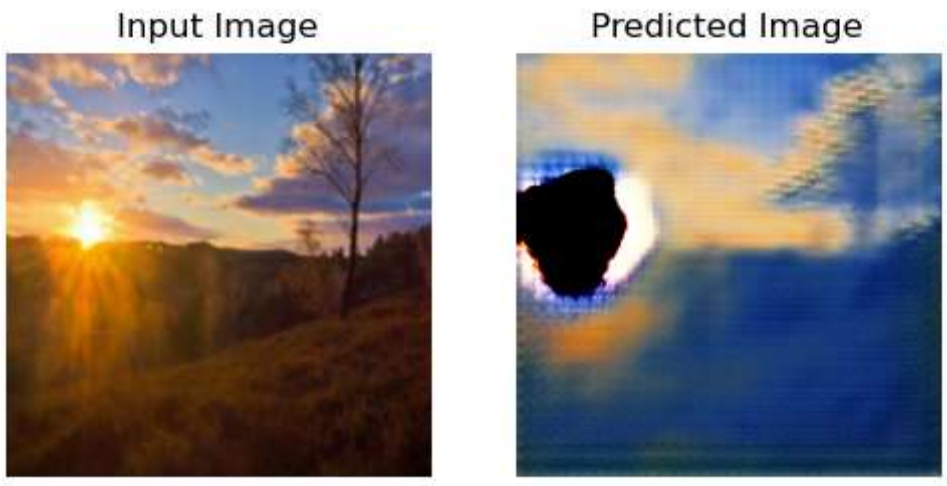
```
.....Time taken for epoch 22 is 49.45908546447754 sec

.....Time taken for epoch 23 is 49.44162201881409 sec

.....Time taken for epoch 24 is 49.15588998794556 sec

.....Time taken for epoch 25 is 49.642492055892944 sec

.....Time taken for epoch 26 is 48.860453367233276 sec

.....Time taken for epoch 27 is 49.60792326927185 sec

.....Time taken for epoch 28 is 49.67510199546814 sec

.....Time taken for epoch 29 is 49.01560091972351 sec

.....Time taken for epoch 30 is 49.45976734161377 sec

.....Time taken for epoch 31 is 49.05631446838379 sec
```



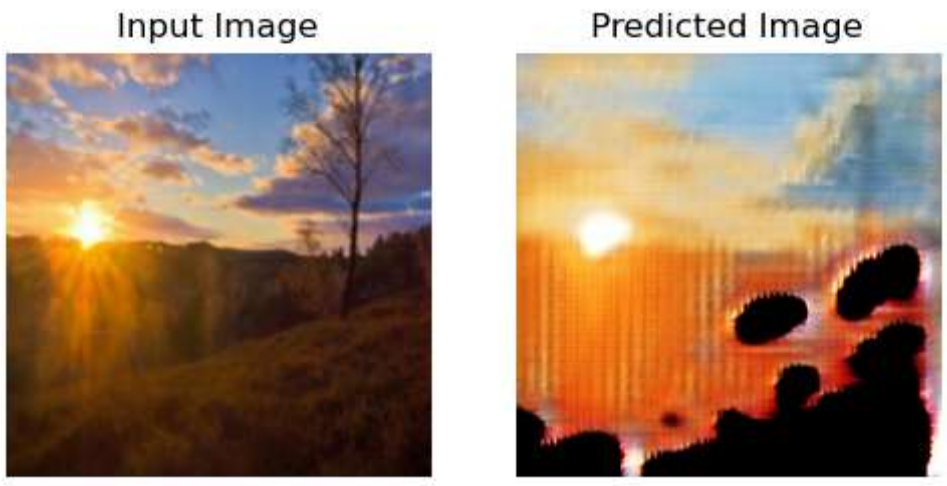
```
.....Time taken for epoch 32 is 49.60667943954468 sec

.....Time taken for epoch 33 is 81.93078374862671 sec

.....Time taken for epoch 34 is 50.21119737625122 sec

.....Time taken for epoch 35 is 50.400036334991455 sec

.....Time taken for epoch 36 is 50.25585961341858 sec

.....Time taken for epoch 37 is 50.496078968048096 sec

.....Time taken for epoch 38 is 49.73182129859924 sec

.....Time taken for epoch 39 is 50.07880401611328 sec

.....Time taken for epoch 40 is 50.0449059009552 sec

.....Time taken for epoch 41 is 49.65529942512512 sec
```



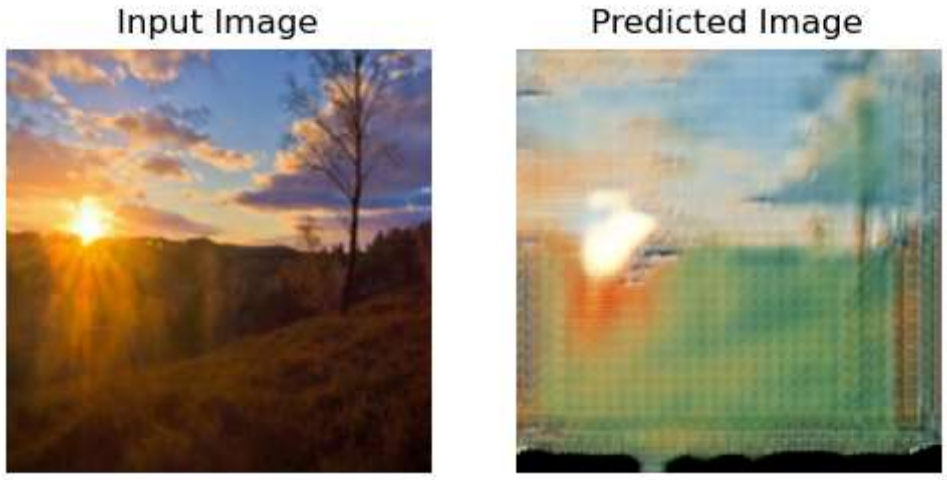
```
.........Time taken for epoch 42 is 50.03731441497803 sec

.........Time taken for epoch 43 is 49.623995542526245 sec

.........Time taken for epoch 44 is 50.1596884727478 sec

.........Time taken for epoch 45 is 49.55078434944153 sec

.........Time taken for epoch 46 is 50.18075132369995 sec

.........Time taken for epoch 47 is 49.94174885749817 sec

.........Time taken for epoch 48 is 49.576645851135254 sec

.........Time taken for epoch 49 is 50.055532455444336 sec

.........Time taken for epoch 50 is 49.556334257125854 sec

.........Time taken for epoch 51 is 49.948631286621094 sec
```



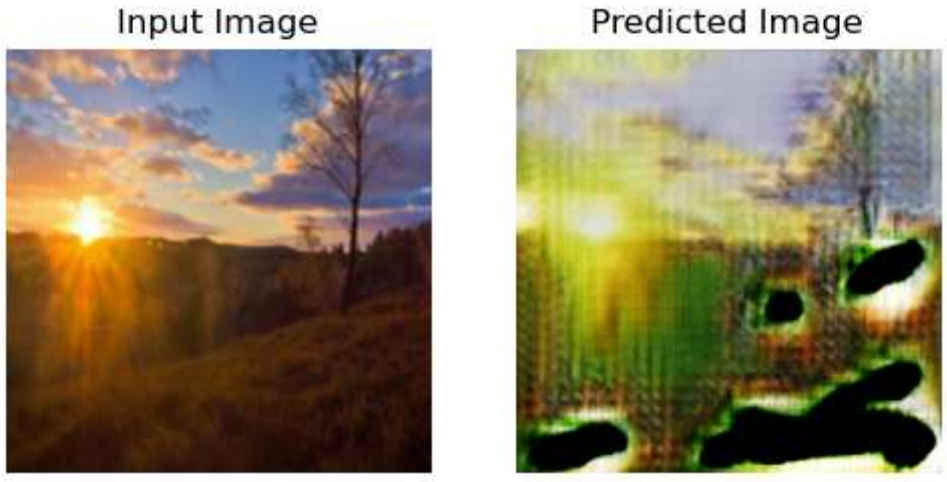
```
.....Time taken for epoch 52 is 81.94206500053406 sec

.....Time taken for epoch 53 is 49.7143394947052 sec

.....Time taken for epoch 54 is 49.868998765945435 sec

.....Time taken for epoch 55 is 49.56581997871399 sec

.....Time taken for epoch 56 is 49.93559455871582 sec

.....Time taken for epoch 57 is 49.54008483886719 sec

.....Time taken for epoch 58 is 49.843169927597046 sec

.....Time taken for epoch 59 is 49.86812472343445 sec

.....Time taken for epoch 60 is 49.6095404624939 sec

.....Time taken for epoch 61 is 49.936655044555664 sec
```



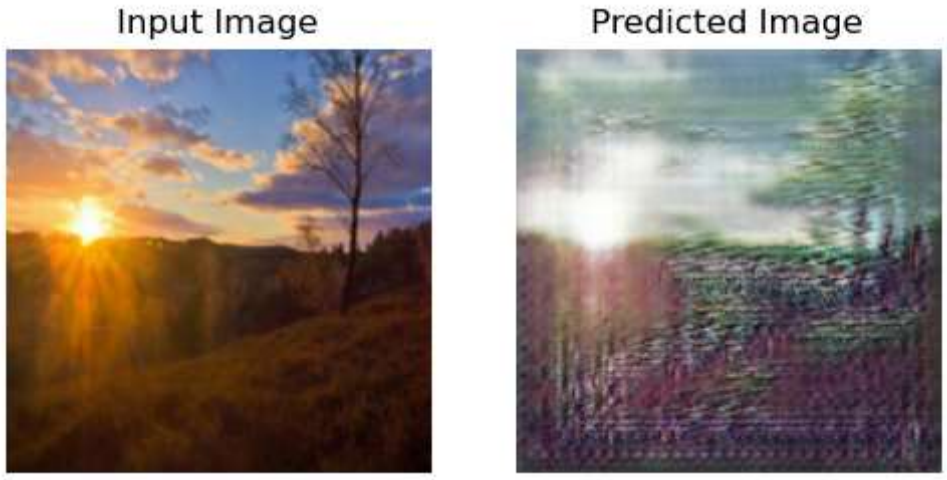
```
.....Time taken for epoch 62 is 81.94157028198242 sec

.....Time taken for epoch 63 is 50.27406358718872 sec

.....Time taken for epoch 64 is 50.17365837097168 sec

.....Time taken for epoch 65 is 49.468013286590576 sec

.....Time taken for epoch 66 is 50.561856746673584 sec

.....Time taken for epoch 67 is 49.3486533164978 sec

.....Time taken for epoch 68 is 50.01993799209595 sec

.....Time taken for epoch 69 is 50.0151047706604 sec

.....Time taken for epoch 70 is 49.99730205535889 sec

.....Time taken for epoch 71 is 49.83262586593628 sec
```



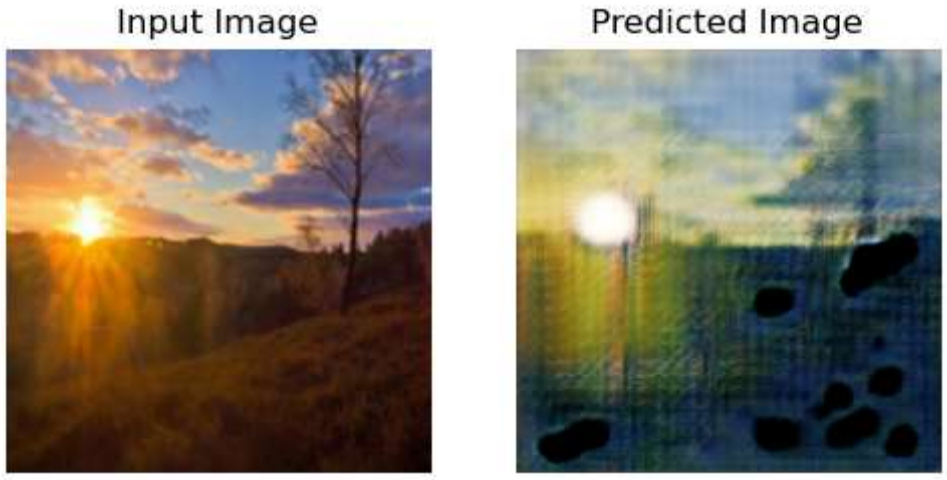
```
.....Time taken for epoch 72 is 49.433735609054565 sec

.....Time taken for epoch 73 is 49.77336573600769 sec

.....Time taken for epoch 74 is 50.28298592567444 sec

.....Time taken for epoch 75 is 50.62512731552124 sec

.....Time taken for epoch 76 is 50.44435739517212 sec

.....Time taken for epoch 77 is 49.69535303115845 sec

.....Time taken for epoch 78 is 50.00769925117493 sec

.....Time taken for epoch 79 is 49.70694708824158 sec

.....Time taken for epoch 80 is 50.113473415374756 sec

.....Time taken for epoch 81 is 49.83724308013916 sec
```



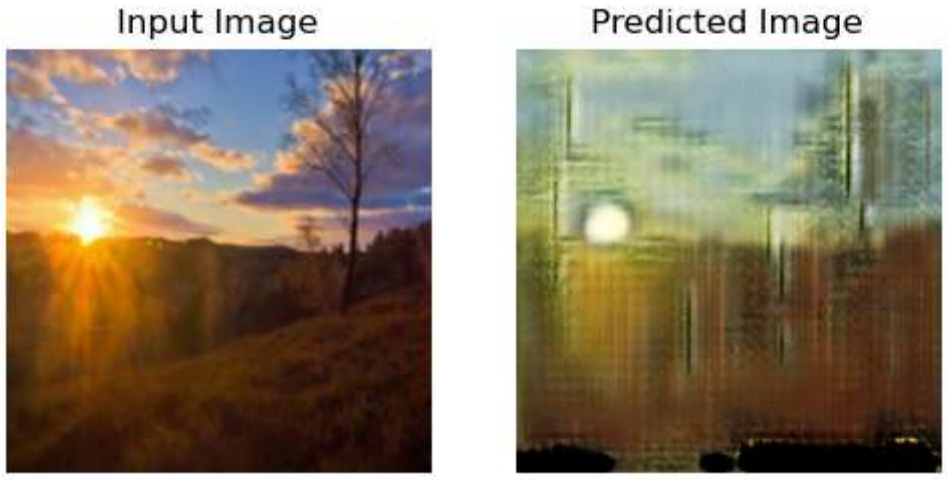
```
.........Time taken for epoch 82 is 50.29565238952637 sec

.........Time taken for epoch 83 is 50.20650029182434 sec

.........Time taken for epoch 84 is 49.74417042732239 sec

.........Time taken for epoch 85 is 50.11090135574341 sec

.........Time taken for epoch 86 is 49.80462956428528 sec

.........Time taken for epoch 87 is 50.2811918258667 sec

.........Time taken for epoch 88 is 50.44066023826599 sec

.........Time taken for epoch 89 is 49.84257626533508 sec

.........Time taken for epoch 90 is 50.394237995147705 sec

.........Time taken for epoch 91 is 50.039215087890625 sec
```



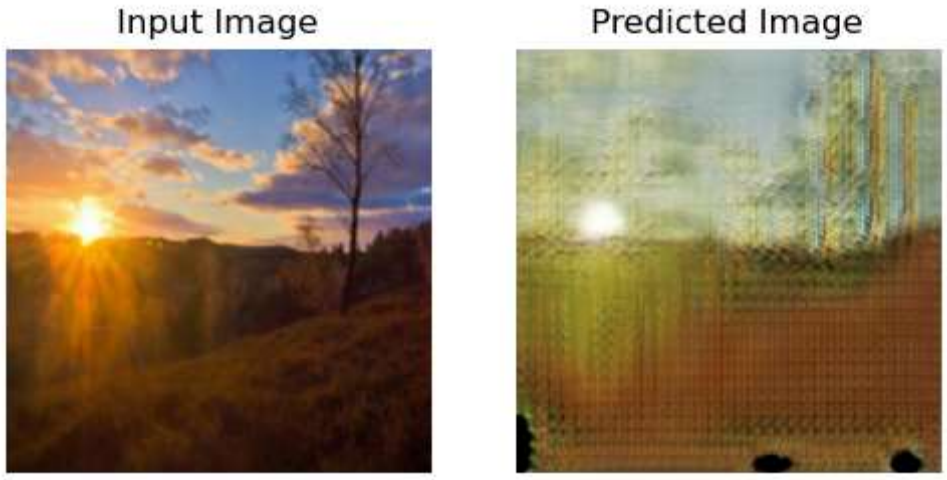
```
.....Time taken for epoch 92 is 50.48321199417114 sec

.....Time taken for epoch 93 is 50.541160583496094 sec

.....Time taken for epoch 94 is 50.176077365875244 sec

.....Time taken for epoch 95 is 50.39259958267212 sec

.....Time taken for epoch 96 is 49.828816413879395 sec

.....Time taken for epoch 97 is 50.152320861816406 sec

.....Time taken for epoch 98 is 50.12226939201355 sec

.....Time taken for epoch 99 is 50.17091155052185 sec

.....Time taken for epoch 100 is 50.286945819854736 sec

.....Time taken for epoch 101 is 49.69802737236023 sec
```



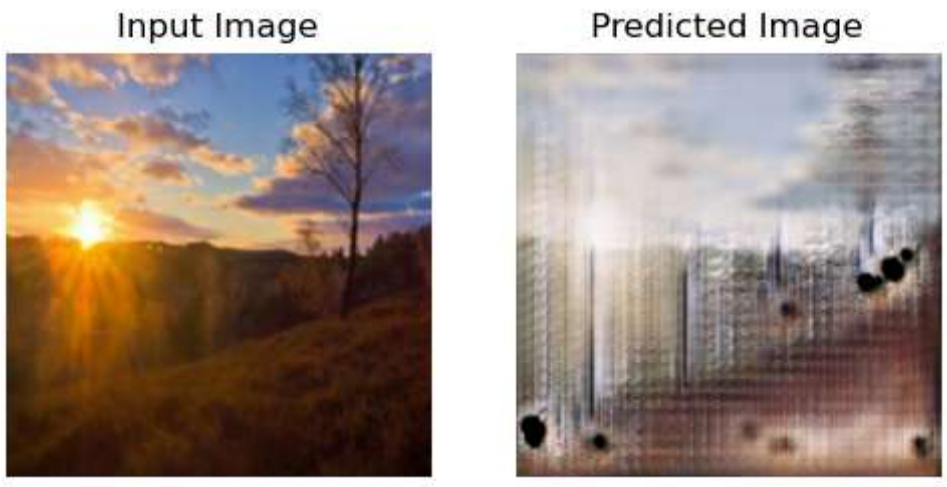
```
..........Time taken for epoch 102 is 50.1968994140625 sec

..........Time taken for epoch 103 is 49.79049062728882 sec

..........Time taken for epoch 104 is 49.853729248046875 sec

..........Time taken for epoch 105 is 49.99528765678406 sec

..........Time taken for epoch 106 is 49.71103286743164 sec

..........Time taken for epoch 107 is 49.816585063934326 sec

..........Time taken for epoch 108 is 49.58667826652527 sec

..........Time taken for epoch 109 is 49.79467511177063 sec

..........Time taken for epoch 110 is 49.37194490432739 sec

..........Time taken for epoch 111 is 49.87623167037964 sec
```



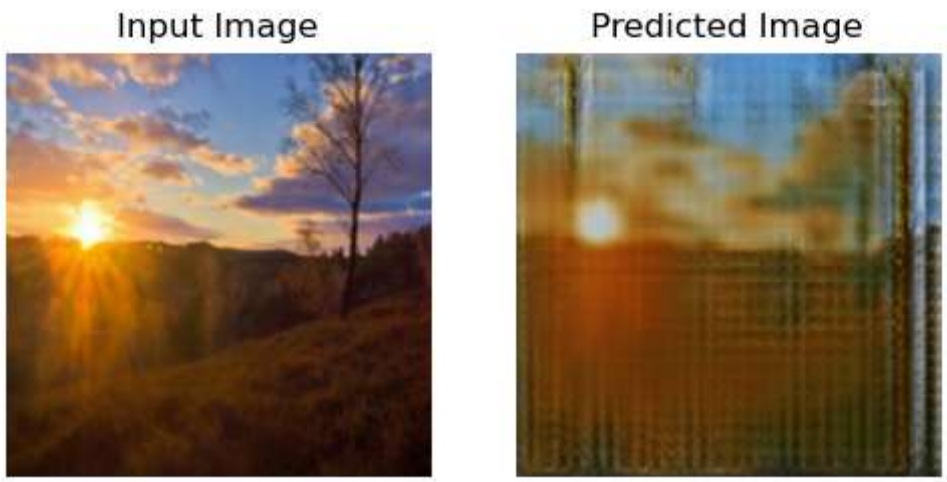
```
.....Time taken for epoch 112 is 50.027092933654785 sec

.....Time taken for epoch 113 is 49.20035743713379 sec

.....Time taken for epoch 114 is 49.849557876586914 sec

.....Time taken for epoch 115 is 49.416460275650024 sec

.....Time taken for epoch 116 is 49.894147872924805 sec

.....Time taken for epoch 117 is 49.4683632850647 sec

.....Time taken for epoch 118 is 50.125732421875 sec

.....Time taken for epoch 119 is 49.987693548202515 sec

.....Time taken for epoch 120 is 49.589043617248535 sec

.....Time taken for epoch 121 is 49.9901909828186 sec
```



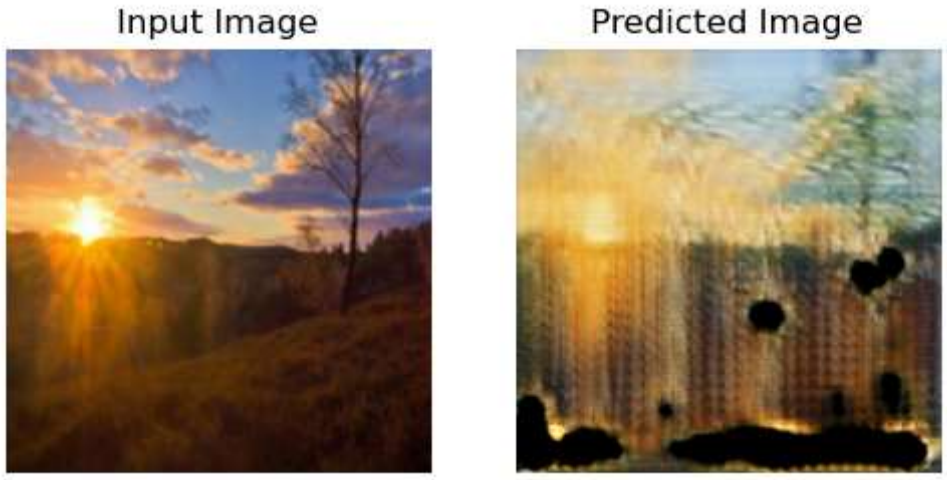
```
.....Time taken for epoch 122 is 49.572205543518066 sec

.....Time taken for epoch 123 is 49.928550481796265 sec

.....Time taken for epoch 124 is 50.125176191329956 sec

.....Time taken for epoch 125 is 50.040385246276855 sec

.....Time taken for epoch 126 is 50.105294704437256 sec

.....Time taken for epoch 127 is 49.83628177642822 sec

.....Time taken for epoch 128 is 50.096606969833374 sec

.....Time taken for epoch 129 is 49.77556228637695 sec

.....Time taken for epoch 130 is 50.26339507102966 sec

.....Time taken for epoch 131 is 50.06994271278381 sec
```



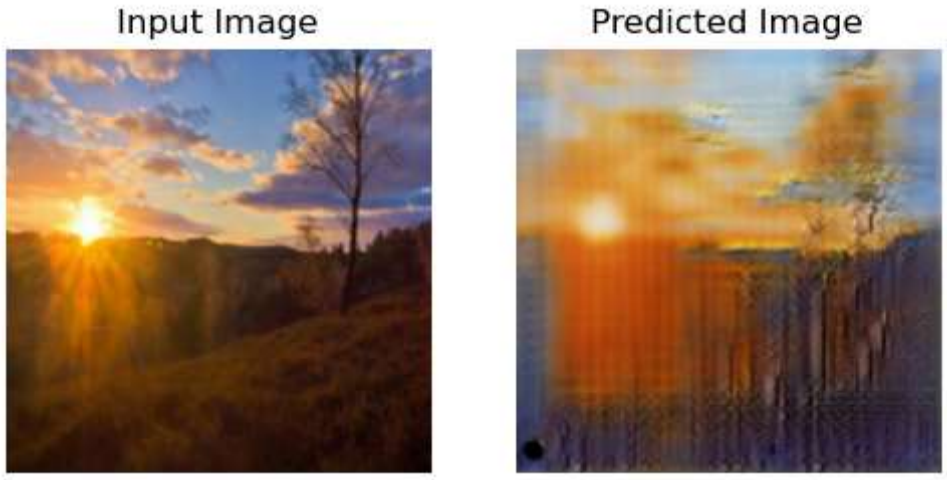
```
.........Time taken for epoch 132 is 49.79015755653381 sec

.........Time taken for epoch 133 is 50.239213943481445 sec

.........Time taken for epoch 134 is 49.75906324386597 sec

.........Time taken for epoch 135 is 50.217275857925415 sec

.........Time taken for epoch 136 is 50.252376317977905 sec

.........Time taken for epoch 137 is 49.87161374092102 sec

.........Time taken for epoch 138 is 50.385621786117554 sec

.........Time taken for epoch 139 is 49.80830669403076 sec

.........Time taken for epoch 140 is 50.188350200653076 sec

.........Time taken for epoch 141 is 49.96118474006653 sec
```



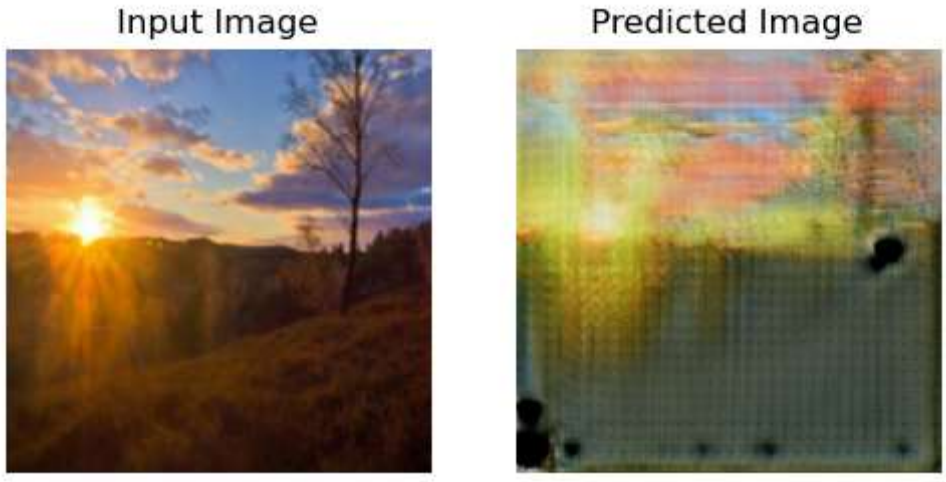
```
.....Time taken for epoch 142 is 49.936546325683594 sec

.....Time taken for epoch 143 is 50.20973992347717 sec

.....Time taken for epoch 144 is 49.8569974899292 sec

.....Time taken for epoch 145 is 50.2376549243927 sec

.....Time taken for epoch 146 is 49.63392996788025 sec

.....Time taken for epoch 147 is 50.101115703582764 sec

.....Time taken for epoch 148 is 50.10811448097229 sec

.....Time taken for epoch 149 is 49.90685224533081 sec

.....Time taken for epoch 150 is 50.250338077545166 sec

.....Time taken for epoch 151 is 49.77262854576111 sec
```



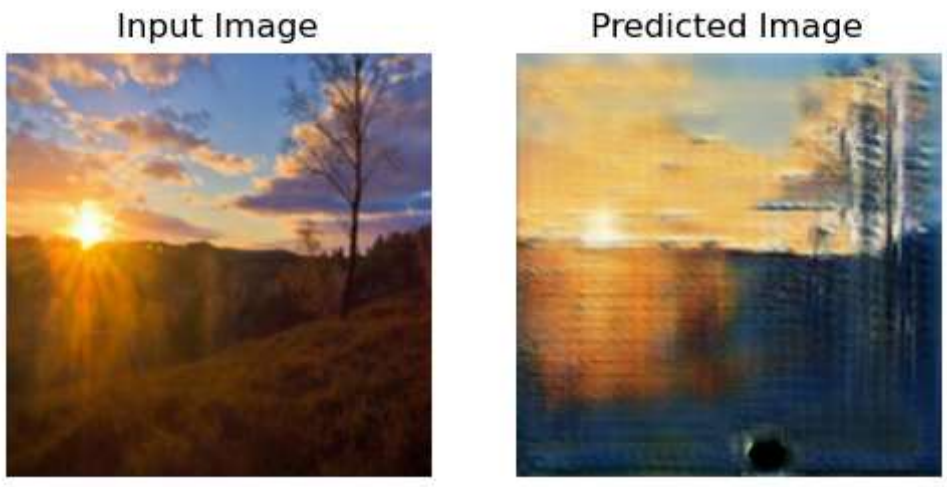
```
.....Time taken for epoch 152 is 50.03738498687744 sec

.....Time taken for epoch 153 is 50.08024024963379 sec

.....Time taken for epoch 154 is 49.82949876785278 sec

.....Time taken for epoch 155 is 50.10397458076477 sec

.....Time taken for epoch 156 is 49.71296048164368 sec

.....Time taken for epoch 157 is 50.139132499694824 sec

.....Time taken for epoch 158 is 49.83730983734131 sec

.....Time taken for epoch 159 is 50.07235050201416 sec

.....Time taken for epoch 160 is 50.03105449676514 sec

.....Time taken for epoch 161 is 49.6344575881958 sec
```



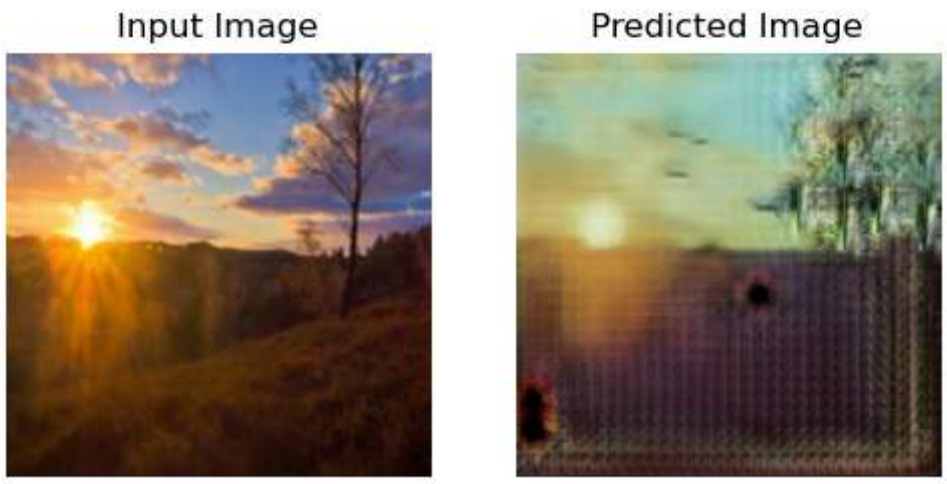
```
.....Time taken for epoch 162 is 50.07298231124878 sec

.....Time taken for epoch 163 is 49.88817310333252 sec

.....Time taken for epoch 164 is 50.00826978683472 sec

.....Time taken for epoch 165 is 81.93554162979126 sec

.....Time taken for epoch 166 is 49.670182943344116 sec

.....Time taken for epoch 167 is 50.1710741519928 sec

.....Time taken for epoch 168 is 49.67395257949829 sec

.....Time taken for epoch 169 is 50.071677684783936 sec

.....Time taken for epoch 170 is 49.71294569969177 sec

.....Time taken for epoch 171 is 50.194865465164185 sec
```



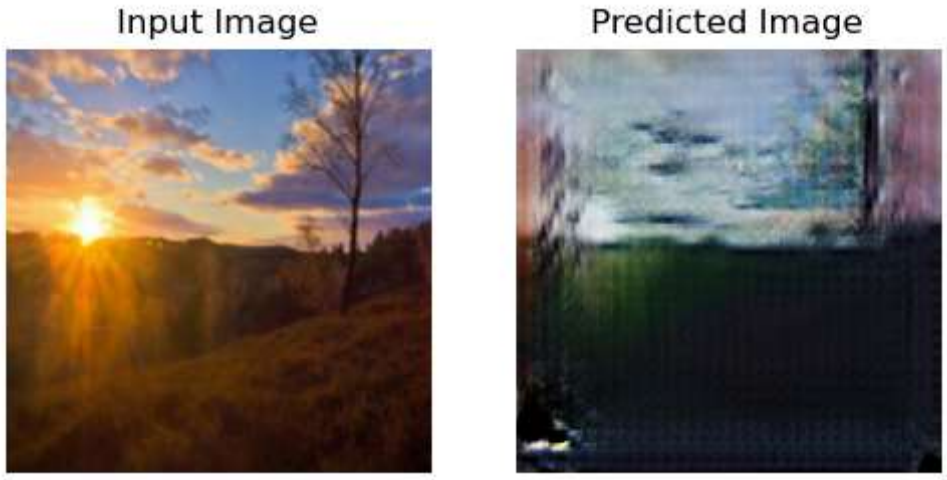
```
.....Time taken for epoch 172 is 50.06757688522339 sec

.....Time taken for epoch 173 is 49.7196729183197 sec

.....Time taken for epoch 174 is 50.135106325149536 sec

.....Time taken for epoch 175 is 49.70363116264343 sec

.....Time taken for epoch 176 is 50.190181732177734 sec

.....Time taken for epoch 177 is 50.21328854560852 sec

.....Time taken for epoch 178 is 49.6876266002655 sec

.....Time taken for epoch 179 is 49.81106090545654 sec

.....Time taken for epoch 180 is 49.32806754112244 sec

.....Time taken for epoch 181 is 50.374022245407104 sec
```



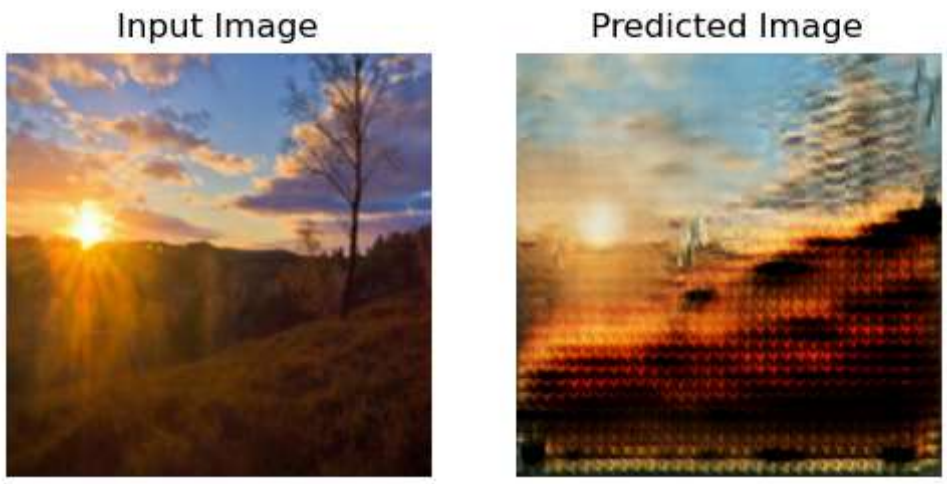
```
.....Time taken for epoch 182 is 49.716400384902954 sec

.....Time taken for epoch 183 is 50.25943374633789 sec

.....Time taken for epoch 184 is 49.61131978034973 sec

.....Time taken for epoch 185 is 49.394386768341064 sec

.....Time taken for epoch 186 is 50.20989680290222 sec

.....Time taken for epoch 187 is 49.804776668548584 sec

.....Time taken for epoch 188 is 50.252723693847656 sec

.....Time taken for epoch 189 is 50.43281865119934 sec

.....Time taken for epoch 190 is 50.12320160865784 sec

.....Time taken for epoch 191 is 50.330721378326416 sec
```



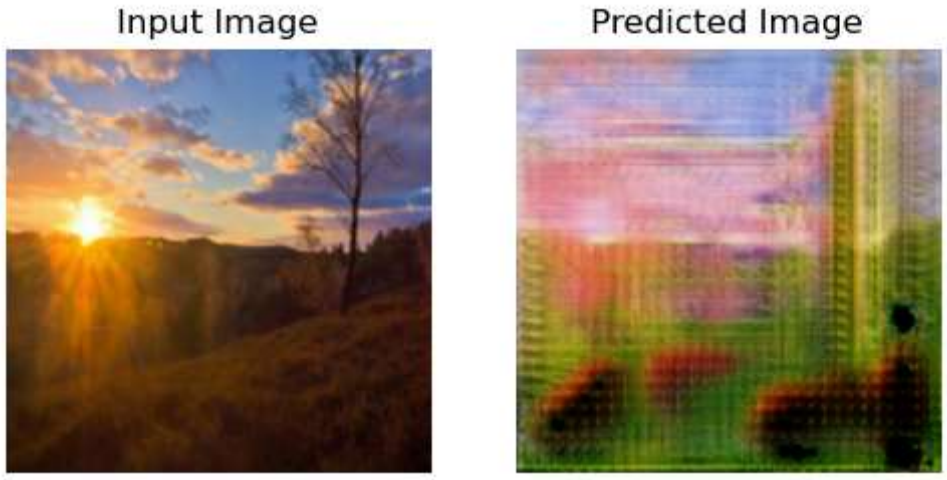
```
.........Time taken for epoch 192 is 49.79965138435364 sec

.........Time taken for epoch 193 is 50.197343826293945 sec

.........Time taken for epoch 194 is 49.76070690155029 sec

.........Time taken for epoch 195 is 50.22727847099304 sec

.........Time taken for epoch 196 is 50.269490480422974 sec

.........Time taken for epoch 197 is 49.75647521018982 sec

.........Time taken for epoch 198 is 50.02175498008728 sec

.........Time taken for epoch 199 is 49.836970806121826 sec

.........Time taken for epoch 200 is 50.099777936935425 sec

.........Time taken for epoch 201 is 50.19626259803772 sec
```



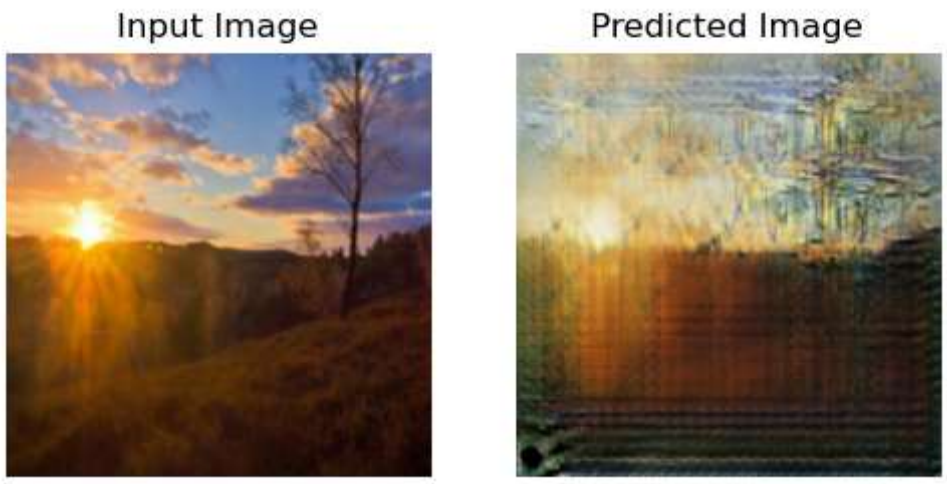
```
.....Time taken for epoch 202 is 49.63320326805115 sec

.....Time taken for epoch 203 is 50.131882190704346 sec

.....Time taken for epoch 204 is 49.68277645111084 sec

.....Time taken for epoch 205 is 50.235591411590576 sec

.....Time taken for epoch 206 is 49.81107783317566 sec

.....Time taken for epoch 207 is 50.10271096229553 sec

.....Time taken for epoch 208 is 50.21267604827881 sec

.....Time taken for epoch 209 is 49.6089653968811 sec

.....Time taken for epoch 210 is 49.99112057685852 sec

.....Time taken for epoch 211 is 49.811115741729736 sec
```



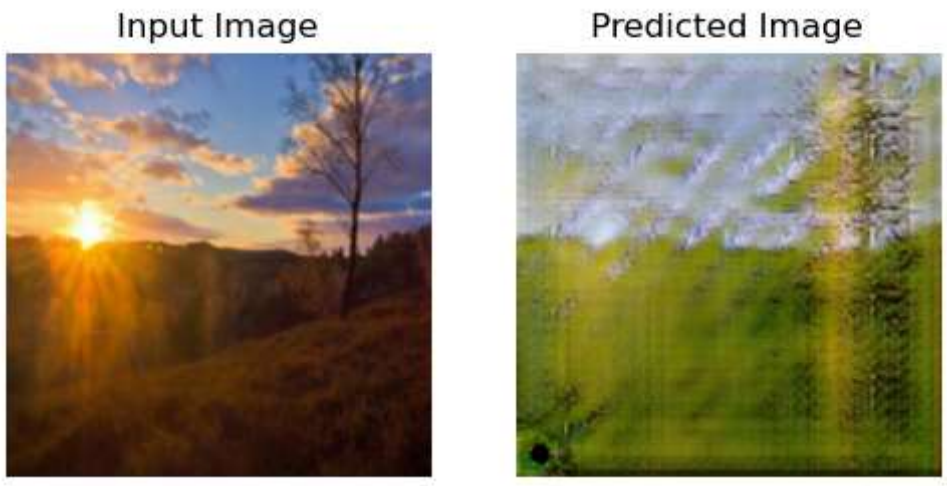
```
.....Time taken for epoch 212 is 50.959097385406494 sec

.....Time taken for epoch 213 is 50.82206964492798 sec

.....Time taken for epoch 214 is 49.880221366882324 sec

.....Time taken for epoch 215 is 81.9388415813446 sec

.....Time taken for epoch 216 is 49.544172286987305 sec

.....Time taken for epoch 217 is 50.077903509140015 sec

.....Time taken for epoch 218 is 81.93012166023254 sec

.....Time taken for epoch 219 is 49.66624140739441 sec

.....Time taken for epoch 220 is 49.855305194854736 sec

.....Time taken for epoch 221 is 49.58433723449707 sec
```



```
..........Time taken for epoch 222 is 49.926191329956055 sec

..........Time taken for epoch 223 is 49.43721413612366 sec

..........Time taken for epoch 224 is 49.94192838668823 sec

..........Time taken for epoch 225 is 50.09064197540283 sec

..........Time taken for epoch 226 is 49.57293915748596 sec

..........Time taken for epoch 227 is 49.96079921722412 sec

..........Time taken for epoch 228 is 49.51835513114929 sec

..........Time taken for epoch 229 is 49.99121356010437 sec

..........Time taken for epoch 230 is 49.55814814567566 sec

..........Time taken for epoch 231 is 49.927253007888794 sec
```



```
.........Time taken for epoch 232 is 49.98669934272766 sec

.........Time taken for epoch 233 is 49.61114168167114 sec

.........Time taken for epoch 234 is 49.849226236343384 sec

.........Time taken for epoch 235 is 49.76348400115967 sec

.........Time taken for epoch 236 is 50.118064165115356 sec

.........Time taken for epoch 237 is 50.10759878158569 sec

.........Time taken for epoch 238 is 49.630699634552 sec

.........Time taken for epoch 239 is 49.939342737197876 sec

.........Time taken for epoch 240 is 49.64512252807617 sec

.........Time taken for epoch 241 is 50.211458683013916 sec
```



```
.....Time taken for epoch 242 is 49.51837730407715 sec

.....Time taken for epoch 243 is 50.14661383628845 sec

.....Time taken for epoch 244 is 50.004823446273804 sec

.....Time taken for epoch 245 is 49.5429892539978 sec

.....Time taken for epoch 246 is 49.93751358985901 sec

.....Time taken for epoch 247 is 49.54403114318848 sec

.....Time taken for epoch 248 is 50.16334295272827 sec

.....Time taken for epoch 249 is 49.471174478530884 sec

.....Time taken for epoch 250 is 50.2634711265564 sec
```

Test the model

Lets feed the model some samples to visualize how it performs. Since our original dataset applied random jittering, we will create a different dataset for visualization and submission where no jittering is applied.

In [27]:

```
#Implement helper functions to minimally process the images
def minprocess_tfrecord(example,jitter=True):
    tfrecord_format = {
        "image_name": tf.io.FixedLenFeature([], tf.string),
        "image": tf.io.FixedLenFeature([], tf.string),
        "target": tf.io.FixedLenFeature([], tf.string)
    }
    example = tf.io.parse_single_example(example, tfrecord_format)
    #decode
    image = tf.image.decode_jpeg(example['image'], channels=3)
    #normalize so colors on the upper end of the RGB spectrum are not washed out
    image = (tf.cast(image, tf.float32)/ 127.5) - 1
    return image

#implement helper function to reload the datasets
def reload_dataset(filenames, labeled=True, ordered=False):
    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.map(minprocess_tfrecord, num_parallel_calls=tf.data.AUTOTUNE)
    return dataset

reloaded_photos=reload_dataset(photo_files)
```

In [29]:

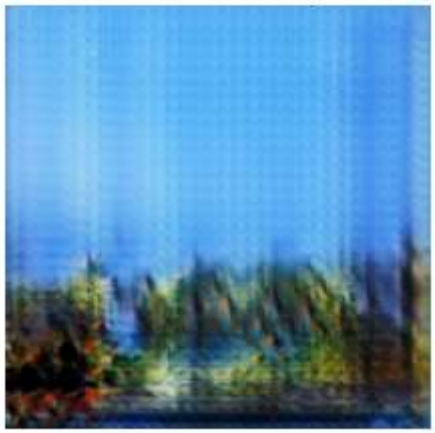
```
def generate_images(model, test_input):
    prediction = model(test_input[None, ...])
    plt.figure(figsize=(6, 6))
    display_list = [test_input, prediction[0]]
    title = ['Input Image', 'Predicted Image']

    for i in range(2):
        plt.subplot(1, 2, i+1)
        plt.title(title[i])
        # getting the pixel values between [0, 1] to plot it.
        plt.imshow(display_list[i] * 0.5 + 0.5)
        plt.axis('off')
    plt.show()

for inp in reloaded_photos.take(10):
    generate_images(generator_m, inp)
```

Input Image

Predicted Image

Input Image

Predicted Image

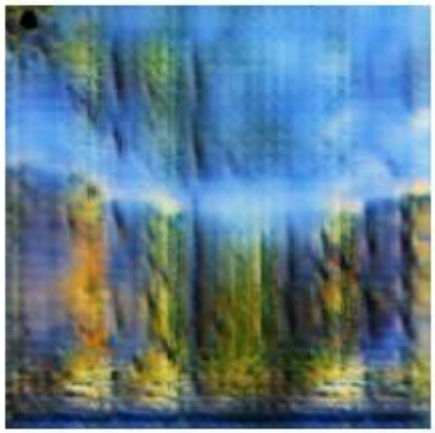


Input Image

Predicted Image

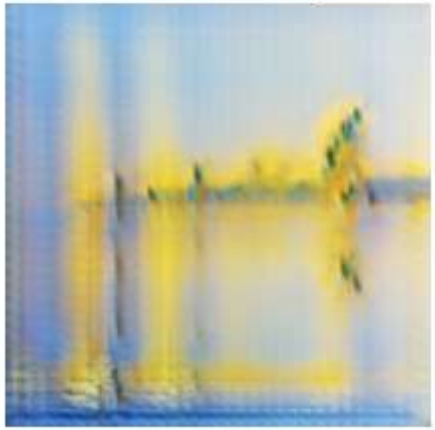
Input Image

Predicted Image



Input Image

Predicted Image



Input Image

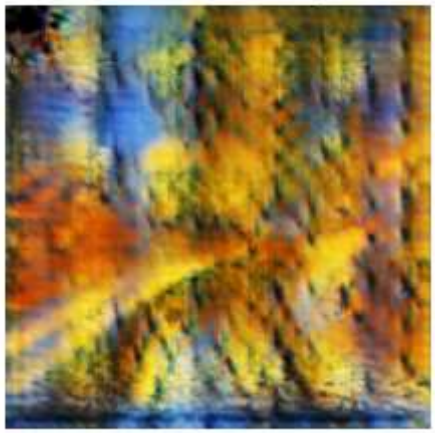
Predicted Image

Input Image

Predicted Image

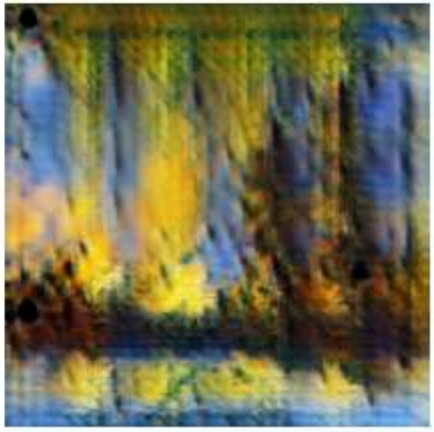
Input Image

Predicted Image



Input Image

Predicted Image



Input Image

Predicted Image

It appears the model is transforming the images, but the results leave a lot to be desired. Photos will be generated and submitted to kaggle for a formal evaluation.

In [41]:

```
! mkdir ../images
i = 1
for img in reloaded_photos:
    prediction = generator_m(img[None,...])[0].numpy()
    prediction = (prediction * 127.5 + 127.5).astype(np.uint8)
    im = PIL.Image.fromarray(prediction)
    im.save("../images/" + str(i) + ".jpg")
    i += 1
shutil.make_archive("/kaggle/working/images", 'zip', "/kaggle/images")
```

Out[41]:

```
'/kaggle/working/images.zip'
```

The final score achieved for the project was

Conclusion

While the model is opperational, the output is less than desirable. This might suggest that Enet architechture is not as well suited for CycleGAN as Unet or Resnet, but further testing would be required. If I were to develop the project further, I would try implememnting an Enet discriminator to see if using a more similiar discriminator effected the overall quality of the image produced. I would also test a hybrid Enet/Unet approach to see if adding skip connections to encoding and decoding layers of the same size helped produce more visually pleasing results. Additionally, I would also store the loss values at every step to to better diagnose the performance at each epoch. Lastly, I would try to figure out a workaround to achieve more epochs while still using a free cloud environment. This could probably be accomplished by exporting the weights and using them as a starting point to intitialize a new model.

Sources

https://www.kaggle.com/code/amyjang/monet-cyclegan-tutorial/notebook (https://www.kaggle.com/code/amyjang/monet-cyclegan-tutorial/notebook)

https://www.tensorflow.org/tutorials/generative/dcgan (https://www.tensorflow.org/tutorials/generative/dcgan)

https://www.tensorflow.org/tutorials/generative/pix2pix (https://www.tensorflow.org/tutorials/generative/pix2pix)

https://www.tensorflow.org/tutorials/generative/cyclegan (https://www.tensorflow.org/tutorials/generative/cyclegan)

https://arxiv.org/pdf/1606.02147.pdf (https://arxiv.org/pdf/1606.02147.pdf)

https://arxiv.org/pdf/1703.10593.pdf (https://arxiv.org/pdf/1703.10593.pdf)

https://arxiv.org/pdf/1611.07004.pdf (https://arxiv.org/pdf/1611.07004.pdf)

https://arxiv.org/pdf/1512.03385.pdf (https://arxiv.org/pdf/1512.03385.pdf)

https://aditi-mittal.medium.com/introduction-to-u-net-and-res-net-for-image-segmentation-9afcb432ee2f (https://aditi-mittal.medium.com/introduction-to-u-net-and-res-net-for-image-segmentation-9afcb432ee2f)

https://arxiv.org/pdf/1909.06840.pdf (https://arxiv.org/pdf/1909.06840.pdf)

In [ ]: